Imitation Learning of an Intelligent Navigation System for Mobile Robots using Reservoir Computing *

Eric A. Antonelo, Benjamin Schrauwen, Dirk Stroobandt
Ghent University
Department of Electronics and Information Systems
Sint Pietersnieuwstraat 41, 9000, Ghent, Belgium
Eric.Antonelo@elis.ugent.be

Abstract

The design of an autonomous navigation system for mobile robots can be a tough task. Noisy sensors, unstructured environments and unpredictability are among the problems which must be overcome. Reservoir Computing (RC) uses a randomly created recurrent neural network (the reservoir) which functions as a temporal kernel of rich dynamics that projects the input to a high dimensional space. This projection is mapped into the desired output (only this mapping must be learned with standard linear regression methods). In this work, RC is used for imitation learning of navigation behaviors generated by an intelligent navigation system in the literature. Obstacle avoidance, exploration and target seeking behaviors are reproduced with an increase in stability and robustness over the original controller. Experiments also show that the system generalizes the behaviors for new environments.

1. Introduction

Autonomous robots form an area of research which is rapidly increasing and becoming very important for our society. Several applications of intelligent robotics are becoming more evident, such as: service and domestic robotics, surveillance, and autonomous operation in hazardous environments.

Traditional approaches in robotics are usually based on the *sense-model-plan-act* framework. They require the modeling of the environment of the robot and also usually present a planning program which generates the complete trajectory a priori. It is easy to see that such an approach presents high computational costs related to path planning routines. Moreover, world modeling may be extremely difficult to accomplish.

New approaches to robotics have been proposed early in the literature [6, 7]. Instead of having several modules for perception, world modeling, planning and execution, the new approach is based on individual intelligent control modules, where each one contributes to behavior generation for a robot. Computational intelligence techniques (e.g., neural networks and evolutionary algorithms) have been the most used techniques for designing such autonomous intelligent systems [1, 5, 8] due to inherent characteristics such as robustness, adaptivity, and learning.

In this work, we are interested in using Recurrent Neural Networks (RNNs) for efficient identification of a particular intelligent navigation system in the literature [1]. This intelligent system learns to navigate in its environment by interacting with it. After some learning period, it is able to generate target seeking and obstacle avoidance behaviors while efficiently exploring its environment. This work will use such a system to generate examples of navigation strategies (or behaviors) that will be used for training a RNN-based controller by an imitation learning process.

Most training algorithms for RNNs have high computational costs and have problems with convergence of the training process. A recently proposed powerful alternative to such traditional RNNs is Reservoir Computing [16]. Reservoir Computing uses a fixed (usually random) RNN that is used as a reservoir of rich dynamics and a linear static readout output layer (see Fig. 1). Only the readout output layer is trained in a supervised way, while the recurrent part of the network (the so called *reservoir*) has fixed weights. The reservoir weights are usually scaled so that its dynamic regime is situated at the edge of stability.

The term Reservoir Computing is a unifying concept for other specific computing techniques: Echo State Networks (ESNs) [9] and Liquid State Machines (LSMs) [13]. Theoretical analysis of reservoir computing methods [10]

*This research is partially funded by FWO Flanders project G.0317.05. Eric A. Antonelo is sponsored by a BOF grant.

1522-4899/08 $25.00 © 2008 IEEE
DOI 10.1109/SBRN.2008.32

and a broad range of applications [16] (which sometimes even drastically outperform the current state-of-the-art [11]) show that RC is very powerful and overcomes many of the problems of traditional RNN training such as slow convergence, bifurcations and high computational requirements.

Reservoir Computing has been successfully applied to a wide range of robotic tasks. In [4], RC is used for complex event detection and robot localization in the context of small mobile robots with few noisy sensors. In that work, two different robot models are used, including the e-puck robot with 8 infra-red sensors. In [3], RC is used in various robotic tasks including prediction of robot coordinates, map learning and path generation. The work in [14] uses an ESN for motor speed control of a differential drive robot, and a LSM is used for imitation learning of simple obstacle avoidance behaviors in [8]. RC has also been used for modeling the road sign problem in [5], where a mobile robot must remember a previously given stimulus (light sign) in order to accomplish a delayed-response task successfully. All these robotic tasks are performed very well by a RC network. The short-term memory in the reservoir makes more complex computation possible, while the training algorithm simply adjusts readout weights by using linear regression methods.

This work shows that this new computing paradigm can be used to efficiently learn navigation strategies from examples given by an (non-linear) intelligent navigation system for mobile robots [1]. After the imitation learning process, the RC network performs very similarly to the original system, being able to generate obstacle avoidance, exploration and target seeking behaviors with an increase in robustness and stability of the robot controller when compared to the original system. The RC network is also able to generalize the navigation skills to new environments.

2. Reservoir computing

This work uses an Echo State Network composed of a discrete hyperbolic-tangent RNN (i.e., the reservoir) and a linear readout output layer which maps the reservoir states to the desired output. The general state update equation for the nodes in the reservoir and the readout output equation are as follows:

$$\mathbf{x}(t+1) = f\left(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{o}}\mathbf{y}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}}\right) \quad (1)$$

$$\mathbf{y}(t+1) = \mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{x}(t+1) + \mathbf{W}^{\mathrm{o}}_{\mathrm{i}}\mathbf{u}(t) + \mathbf{W}^{\mathrm{o}}_{\mathrm{o}}\mathbf{y}(t) + \mathbf{W}^{\mathrm{o}}_{\mathrm{b}} \quad (2)$$

where: $\mathbf{u}(t)$ denotes the input at time t; $\mathbf{x}(t)$ represents the reservoir state; $\mathbf{y}(t)$ is the output; and $f() = \tanh()$ is the hyperbolic tangent activation function. The weight matrices $\mathbf{W}$ represent the connections between the nodes of the network (where r, i, o, b denotes $reservoir, input, output$, and $bias$, respectively). All weight matrices to the reservoir (denoted as $\mathbf{W}^{\mathrm{r}}_{\cdot}$) are initialized randomly (represented by solid arrows in Fig. 1), while all connections to the output (denoted as $\mathbf{W}^{\mathrm{o}}_{\cdot}$) are trained (represented by dashed arrows in Fig. 1). The initial state is set to $\mathbf{x}(0) = \mathbf{0}$.

However, for the experiments in this work, we discard the output feedback to the reservoir and we add a leak rate α as in [15] to the state update equation:

$$\begin{aligned}\mathbf{x}(t+1) \quad = \quad & f((1-\alpha)\mathbf{x}(t) + \\ & + \alpha(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}(t) + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}})). \quad (3)\end{aligned}$$

The output calculation gets simpler once we do not use the direct connections from input to output neither the connections from output to output:

$$\mathbf{y}(t+1) = \mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{x}(t+1) + \mathbf{W}^{\mathrm{o}}_{\mathrm{b}}. \quad (4)$$

The leak rate can effectively tune the timescale of the dynamics of the reservoir. If the leak rate is chosen correctly, the reservoir dynamics can be adjusted to match the timescale of the input, making it possible to achieve improved performance (this can also be achieved by resampling the input [15, 4]). In this work, some experiments use 2 pools of neurons in the reservoir with distinct leak rates to achieve better performance. Further investigation about timescales in reservoirs and leaky integrator neurons can be found in [15, 12]. Each element of the connection matrix $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$ is drawn from a normal distribution with mean 0 and variance 1. For most applications, the best performance is attained with a reservoir that operates at the edge of stability. The randomly created $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$ matrix is rescaled such that the system is stable. This can be accomplished by rescaling the matrix so that the spectral radius (the largest absolute eigenvalue) of the linearized system is slightly smaller than one [10]. In this work we scale all reservoirs to a spectral radius of $|\lambda_{max}| = 0.9$ which is near optimal for most experiments, but the value of the spectral radius could be further optimized for each experiment separately.

Training is performed using linear regression (least squares method). The computational efforts for training are related to computing a matrix product and inversion. It takes just a few seconds to train a RC network for the experiments in this work on an Intel Core2 Duo processor-based

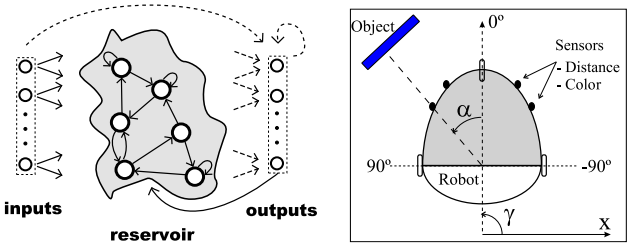


Figure 1. Reservoir Computing network (left) and Robot model (right).

system. Once trained, the resulting RC-based system can be used for real-time operation on moderate hardware since the computations are very fast (only matrix multiplications of small matrices).

The Normalized Mean Square Error (NMSE) is used as a performance measure in this work and is defined as:

$$\mathrm{NMSE} = \frac{\langle (y_d - y)^2 \rangle}{\sigma^2_{y_d}} \quad (5)$$

where the numerator is the mean square error of the output y and the denominator is the variance of desired output y_d.

3. Robot Model

We use a robot model that is part of the 2D SINAR simulator [1] in the following experiments. This simulation environment generates the data necessary for training the RC networks. The environment of the robot is composed of several objects, each one of a particular color. Obstacles (repulsive objects) have the blue color whereas targets (attractive objects) have the yellow color. The robot model is shown in Fig. 1. The robot interacts with the environment by distance and color sensors; and by one actuator which controls the movement direction (turning). Seventeen (17) sensor positions are distributed uniformly over the front of the robot (from -90° to +90°). Each position holds two virtual sensors (for distance and color perception) [1]. The distance sensors are limited in range (i.e., they saturate for distances greater than 300 distance units (d.u.)) and are noisy (they exhibit Gaussian noise on their readings, generated from $N(0, 60)$ in d.u.). A value of 0 means near some object and a value of 1 means far or nothing detected. At each iteration the robot is able to execute a direction adjustment to the left or to the right in the range [0, 15] degrees and the speed is constant (0.28 distance units (d.u.)/s) (summary in Table 1).

The controller for the SINAR model (based on [1]) is an intelligent navigation system composed of hierarchical neural networks which learn by classical reinforcement learning algorithms. The system learns to seek targets and avoid obstacles as the robot interacts with the environment (by colliding against obstacles and by capturing targets in the environment). It also learns to distinguish targets and obstacles (which present distinct colors) by associating their respective colors to attraction or repulsion behaviors (see [2, 1]). From now on, this controller will be called INASY (Intelligent autonomous NAvigation SYStem).

The INASY controller will provide examples of navigation trajectories to a RC-based robot controller which will be called RECNA (REservoir Computing NAvigation system) from now on. The samples collected from the INASY controller (distance and color sensors, and actuators) are used to train the RECNA controller in a Matlab environment using the RCT Toolbox[1] [16].

4. Identification of Navigation System

4.1. Introduction

This section elaborates on the identification of a nonlinear system for autonomous navigation of mobile robots. As stated in the previous section, the RECNA controller is composed of a RC network which learns by imitation learning of navigation examples given by the INASY controller [1]. The experiments are divided in two groups. The first group of experiments investigates the collision avoidance and exploration behaviors of the RECNA controller (Section 4.2). The second group analyses the target seeking behavior learned by RECNA controller (Section 4.3). Comparisons with the original INASY controller are also made.

In the following, the parameter configuration for the RC network (of the RECNA controller) is presented. The inputs to the network are the instantaneous values of the 17 distance sensors and 17 color sensors. The reservoir size is either 400 or 600 neurons. Table 2 shows 3 different models for the RECNA controller. The models RECNA2 and RECNA3 have 2 pools of neurons with distinct leak rates (that is, half of the neurons in the reservoir uses $\alpha = 0.1$ and the other half uses $\alpha = 1$). These multiple timescales in the reservoir generated by different leak rates can improve performance of the controller, depending on the robotic task which is addressed. The readout layer has 1 output unit which corresponds to the turning (direction adjustment) robot actuator (the robot has constant velocity). The connection matrix from input to the reservoir ($\mathbf{W}^{\mathrm{r}}_{\mathrm{i}}$) is initialized to -0.2, 0.2 and 0 with probabilities 0.1, 0.1 and 0.8, re-

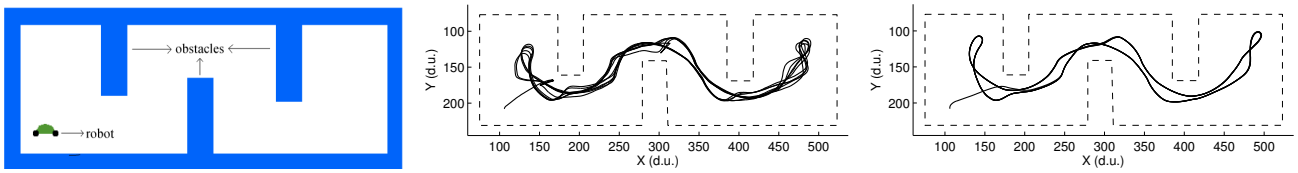
[1]This is an open-source Matlab toolbox for Reservoir Computing which is freely available at http://www.elis.ugent.be/rct

Table 1. Robot model

No. Dist. Sensors	17
No. Color Sensors	17
Range of Dist. Sens.	300 d.u.
Noise on sensors	N(0,60 d.u.)
Speed	0.28 d.u.

Table 2. Parameter configuration

Controller	Reservoir Size	Leak rates (α)
RECNA1	400	1
RECNA2	400	0.1, 1
RECNA3	600	0.1, 1



(a) Environment E1 (b) Trajectory of INASY controller (c) Trajectory of RECNA controller

Figure 2. Environment E1 and respective trajectory of controllers.

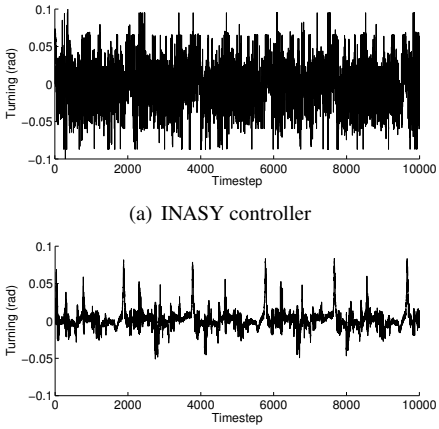
spectively. This parameter setting for weight matrices is not critical for the experiments. Gaussian noise ($N(0, 10^{-6})$) is added to the state update equation (3) during learning for making the reservoir robust to noise.

4.2. Collision Avoidance and Exploration

The first experiment is accomplished using environment E1 (see Fig. 2(a)). The environment is composed of a long corridor with three obstacles. The experiment is executed in three stages. First, the INASY controller learns to navigate in this environment (that is, to avoid obstacles and to explore the environment) (see Fig. 2(b)). In the second stage, data (samples of the sensors and actuators) are collected from the trained INASY controller during a robot run which lasts 30.000 timesteps. The third stage corresponds to training the RECNA controller with the data collected in the second stage by supervised imitation learning (using RECNA1 configuration in Table 2).

The resulting RECNA controller learned to navigate and explore the environment very well. Fig. 2 shows the trajectories given by INASY and RECNA controllers during 10.000 timesteps. We can note that both trajectories are very similar whereas the RECNA controller is more stable and robust to noisy sensors than INASY controller for this particular case (note that the trajectory given by RECNA is thinner and more regular than the other trajectory). Their respective robot actuators (turning) are shown in Fig. 3. It is also interesting to observe that the output of the RECNA controller (Fig. 3(b)) is much less noisy than the output of the INASY controller (Fig. 3(a)). We tested both controllers for 100.000 timesteps in environment E1 and recorded the number of collisions for each run. While the INASY controller had shown 8 collisions, the RECNA controller did not collide at all. Furthermore, we also tested 10 different, randomly created RC networks (trained with the same dataset) in environment E1. Each one of the 10 resulting controllers presented no collisions against obstacles during a test run of 10.000 timesteps.

In order to test the generalization capabilities of the RECNA controller, we trained it on the dataset generated from environment E1 (using RECNA1 configuration) and tested it on a new environment E2 (see Fig. 4). The new environment has corridors of different dimensions and several obstacles which reduce the robot's passageway in the environment. We can observe in Fig. 4 that the trajectory given by RECNA controller in the new environment E2 shows that the controller generalized its navigation capabilities for unknown environments very well. It is able to explore environment E2 avoiding collisions against most of the obstacles. Statistics for this environment are shown in Table 3, consid-



(a) INASY controller

(b) RECNA controller

Figure 3. Output of controllers in E1.

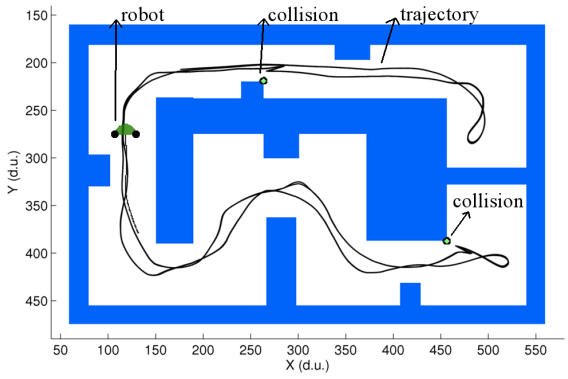


Figure 4. Environment E2 and trajectory of RECNA controller.

Table 3. Results - Environments E1 and E2

Controller	Mean (SD) Collisions Test performance E2	Mean (SD) NMSE Training perf. E1
RECNA1	6.3 (7.66)	0.7855 (0.0017)
RECNA2	4.7 (3.8)	0.7735 (0.0021)
RECNA3	3.5 (3.9)	0.7582 (0.0015)

ering 3 different configurations for the RECNA controller which is trained using environment E1 (see Table 2). It presents the mean and standard deviation (SD) of the number of collisions during 10.000 timesteps for 10 different RC networks (test performance in E2). It also shows the mean and SD of the respective training NMSE (Normalized Mean Square Error) (training performance in E1). The usual RECNA1 controller with 400 neurons and no different leak rates shows an average of 6.3 collisions in environment E2 during 10.000 timesteps. When distinct leaks are considered in the same reservoir (RECNA2), the mean of collisions drops to 4.7 (see Table 3). The performance is further improved to 3.5 collisions in average when the reservoir is made bigger (600 neurons) while also considering distinct leak rates (RECNA3). We can conclude that the existence of multiple timescales in a single reservoir enhances the performance and stability of the controller for unknown environments (bigger reservoirs also yield improvement in performance).

4.3. Target Seeking

Next we investigate the target seeking behavior of RECNA controllers. For this, we also divide the experiments in three stages as in previous section: data generation by pre-trained INASY controllers; training RC-based robot controllers (RECNA) with collected data (using RECNA1 configuration from Table 2); and testing of the resulting RECNA controllers in real-time in the simulator.

The first experiment uses environment E3 (Fig 5). It is a large environment similar to E1, but with 8 targets located in the corners (targets are striped in the figure for clarification). Initially, all targets are located in the environment. Every time the robot captures a target (by just approaching it close enough), it is removed from the environment. When the last target has been captured, all the others are put back in their respective locations.

The INASY controller learns to navigate in the environment and capture all targets repeatedly. The samples generated by this controller are recorded and used for training the RECNA controller by imitation learning. An example of trajectory generated by the resulting RECNA controller is shown in Fig. 5. It is possible to observe that the trajectory tends towards targets, that is, the robot always captures

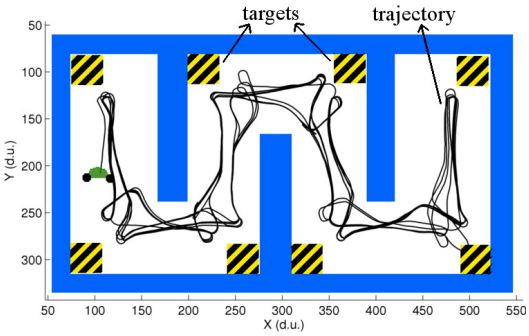
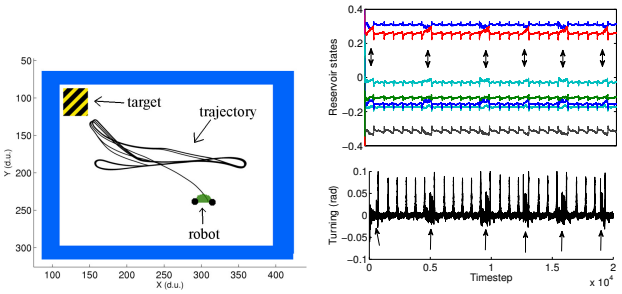


Figure 5. Environment E3 and trajectory of RECNA controller.

Table 4. Results - Environment E3

Controller	No. Collisions	No. Captures
INASY	4	186
RECNA	2	182

the targets if they are present in the environment. The collision avoidance and exploration behaviors are also learned: the robot visits the whole environment with only 2 collisions. Table 4 shows the results for INASY and RECNA controllers in environment E3 for 100.000 timesteps. We can observe that both controllers are very proficient in the considered task. The next experiment investigates the target seeking behavior acquired by the RECNA system. For this, we use environment E4 (see Fig. 6(a)) which contains a target at its upper-left corner. This target is removed from the environment when the robot captures it. After some delay period (3000 or 4000 timesteps) it is put back in its original location. So, the target is constantly reappearing in the environment. The trajectory of the RECNA controller (trained with the previous dataset from environment E3) is



(a) Environment E4 and trajectory of RECNA controller

(b) Reservoir states (upper plot) and turning actuator (lower plot)

Figure 6. Environment E4 and respective results for target seeking.

shown in Fig 6(a). The part of the trajectory which forms a stretched 8 corresponds to a collision avoidance behavior (when the target is not visible). As soon as the target reappears in the environment and the robot localizes it, we can see that the trajectory is modified towards the target area.

The first plot of Fig. 6(b) shows seven randomly selected reservoir states (i.e., activation signals of 7 neurons in the reservoir). The arrows in the figure indicate the moments in which the target appears in the environment. During these time intervals, there are clear changes in the activation signals of the neurons, indicating that a target is present in the environment. The corresponding robot actuator (output of RECNA controller) is shown in the lower plot from Fig. 6(b). The arrows show the instants during which a target seeking behavior is enabled.

5. Conclusion

In this work, we use an intelligent autonomous navigation system for mobile robots [1] for generation of examples of navigation behaviors (such as obstacle avoidance, exploration and target seeking behaviors) which are used to train a Reservoir Computing (RC) network by a imitation learning process. This RC network is composed of a fixed recurrent neural network (the reservoir) and a trainable readout output layer. The learning is done by linear regression which guarantees convergence of the training process in a short time period.

After the imitation learning process, the RC network is able to generate suitable behaviors in several environments in the same way as the original controller. In other words, the non-linear intelligent system [1] is efficiently identified by a RC network which is able to generalize the navigation capabilities to new environments, while having better stability and robustness than the original robot controller. The reservoir, with its short-term memory capabilities, and a simple learning algorithm make non-linear system identification a very effective process.

The importance of different timescales in reservoirs is also demonstrated in this work. The RC-based robot controller performs better (i.e., it shows less collisions) when more than one timescale is present in the reservoir (in the form of leak rates). Future work includes the study of how many behaviors a single RC network can learn. In addition, we could investigate the generation of primitive behaviors by reservoirs which can be combined to generate more complex behavior. Such approach yields applications in control of robots with many degrees of freedom.

References

[1] E. A. Antonelo, A.-J. Baerlvedt, T. Rognvaldsson, and M. Figueiredo. Modular neural network and classical reinforcement learning for autonomous robot navigation: Inhibiting undesirable behaviors. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN)*, pages 498– 505, Vancouver, 2006.

[2] E. A. Antonelo, M. Figueiredo, A.-J. Baerlvedt, and R. Calvo. Intelligent autonomous navigation for mobile robots: spatial concept acquisition and object discrimination. In *Proc. IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, pages 553–557, Helsinki, 2005.

[3] E. A. Antonelo, B. Schrauwen, and J. V. Campenhout. Generative modeling of autonomous robots and their environments using reservoir computing. *Neural Processing Letters*, 26(3):233–249, 2007.

[4] E. A. Antonelo, B. Schrauwen, and D. Stroobandt. Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, 2008.

[5] E. A. Antonelo, B. Schrauwen, and D. Stroobandt. Mobile robot control in the road sign problem using reservoir computing networks. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2008.

[6] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, Mar 1988.

[7] R. Brooks. New approaches to robotics. *Science*, 253:1227–1232, 1991.

[8] H. Burgsteiner. Imitation learning with spiking neural networks and real-world devices. *Int. Journal of Intelligent Real-Time Automation*, 19:741–752, 2006.

[9] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[10] H. Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, 2001.

[11] H. Jaeger and H. Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 308:78–80, April 2 2004.

[12] H. Jaeger, M. Lukosevicius, and D. Popovici. Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20:335–352, 2007.

[13] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[14] M. Salmen and P. G. Plöger. Echo state networks used for motor control. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1953–1958, 2005.

[15] B. Schrauwen, J. Defour, D. Verstraeten, and J. Van Campenhout. The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, 2007.

[16] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. A unifying comparison of reservoir computing methods. *Neural Networks*, 20:391–403, 2007.